*COMP 410 Fall 2015*

Midterm Exam

This exam is closed book, notes, calculators, cell phones, classmates, everything but your own brain. You have 75 minutes to complete the exam. Do all your work on these exam pages. Please sign here (and print your name under it) pledging that the work you submit is your own:

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name (print): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Problem 1 (10%)**

Consider these axioms defining the ADT STI ( Stack of int ):

 datatype STI = New | Push of STI \* int ;

 fun size (New) = 0

 | size (Push(S,i)) = size(S)+1;

 fun pop (New) = New

 | pop (Push(S,i)) = S ;

 fun empty (New) = true

 | empty (Push(S,i)) = false ;

 exception topEmptyStack;

 fun top (New) = raise topEmptyStack

 | top (Push(S,i)) = i ;

We want to add to our ADT the operations “findMax” with this signature:

 findMax: STI 🡪 int

Operation findMax will return the largest integer value in the stack.

Write axioms that will define this new operation.

 **exception maxEmptyStack;**

 **fun findMax ( New ) = raise maxEmptyStack;**

 **fun findMax ( push(S,i) ) = if (empty(S))**

 **then i**

 **else if (findMax(S)<i)**

 **then i**

 **else findMax(S);**

**Problem 2 (15%)**

a) For the following minimum binary heap (drawn as a complete binary tree) fill the items into the array below in the correct positions for the way a binary heap is normally represented with an array.

 3

 . .

 . .

 . .

 12 7

 . . . .

 . . . .

 . . . .

 15 18 9 21

 . . . . . .

 . . . . . .

 27 16 21 52 13 43

 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

 | | | | | | | | | | | | | | | | | |

 array: |\_\_\_|\_3\_|12\_|\_7\_|15\_|18\_|\_9\_|21\_|27\_|16\_|21\_|52\_|13\_|43\_|\_\_\_|\_\_\_|\_\_\_|

 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

b) Using this array representation, show the subscript calculations needed to find the children of the node with value "18" 18 is in slot 5. LC of slot 5 is at 2\*5 = 10 slot

 RC of slot 5 is at (2\*5)+1 = 11 slot

c) Using this array representation, show the subscript calculations needed to find the parent of the node with value "16" 16 is in slot 9. Parent is at floor(9/2) = floor(4.5) = 4 slot

d) Starting with the heap shown, draw the heap (tree representation) that results after two deleteMin operations.

 9

 . .

 . .

 . .

 12 13

 . . . .

 . . . .

 . . . .

 15 18 43 21

 . . . .

 . . . .

 . . . .

 27 16 21 52

**Problem 3 (20%)**

1. Starting with an initially empty Binary Search Tree, show the tree that results after inserting the following values in the order given left to right:

 *6, 13, 8, 4, 19, 11, 5, 9, 12, 7, 2*

 6

 . .

 . .

 . .

 4 13

 . . . .

 . . . .

 . . . .

 2 5 8 19

 . .

 . .

 . .

 7 11

 . .

 . .

 . .

 9 12

1. Starting with the result from (a), show the tree after a delete on 8.

EITHER 6 OR 6

 . . . .

 . . . .

 4 13 4 13

 . . . . . . . .

 . . . . . . . .

 2 5 9 19 2 5 7 19

 . . .

 . . .

 7 11 11

 . . .

 . . .

 12 9 12

1. List the node values in the order produced by a post-order traversal (for the tree in part a)

 2, 5, 4, 7, 9, 12, 11, 8, 19, 13, 6

1. List the node values in the order produced by an pre-order traversal (for the tree in part a)

 6, 4, 2, 5, 13, 8, 7, 11, 9, 12, 19

**Problem 4 (10%)**

For the following items in the order given (left to right) construct a minimum binary heap. Use the O(N) build algorithm we have discussed (not N successive inserts). Show your steps for partial credit. Tree representation is all we need here. Don’t worry about showing the array representation.

 21, 6, 3, 29, 7, 2, 17, 19, 8, 4

 2

 . .

 . .

 4 3

 . . . .

 . . . .

 8 6 21 17

 . . .

 . . .

 19 29 7

**Problem 5 (15%)**

A friend at NC State tells you that using a Binary Search Tree is a good way to sort a collection of N numbers into ascending order. The approach is to build a BST from the N numbers, and then perform an in-order traversal of the tree to get the sorted sequence.

1. What is the worst-case big-Oh complexity of building the BST from the N numbers? **O(N^2) since it is N elements to add and each has worst case O(N)**

 **Also accepted O(N log N) due to error on study guide, although that is the avg case not the worst case**

1. What is the worst-case big-Oh complexity of doing the in-order traversal of the BST once you have it built?

 **O(N)**

1. What is the worst case big-Oh complexity of the whole BST-sort algorithm? **O(N^2) + O(N) is O(N^2) also accepted (N log N) due to error**

**Problem 6 (15%)**

Give a tight big-Oh run-time analysis in terms of N for each of the following code fragments.

 a) public static long A (int N) {

 if (N <= 1) return 1; **ANSWER: O(2^N)**

 return N + A(N-1) + A(N-2);

 }

 b) sum = 0; **ANSWER: O(N^2)**

 for (int i = 0; i < N; i++)

 for (int j = 0; j < i; j++) sum = sum \* 2;

 c) sum = 0;

 for (int i = 0; i < N; i++) **ANSWER:** **O(N^4)**

 for (int j = 0; j < i; j++)

 for (int k=0; k<j\*j; k++) sum = i\*j\*k;

**Problem 7 (15%)**

Let’s play a number guessing game. I am thinking of a number between 0 and 1000 (inclusive of each). You take a guess at it. If your guess is wrong, I will say “no it is higher” or “no it is lower”. Since you were paying attention in class, you know that binary search is a good way to do this. However, a friend from Duke suggests to you the following algorithm. You decide to analyze it and see how good it is.

Let’s say you decide to guess my number this way… starting at 0 you ask “I guess 0”. If I say “no, higher” you say “I guess 5”. If I say “no, higher” you say “I guess 10”. You keep going up to the next multiple of 5 until I say “no, lower” (or, of couse, “yes that’s it”). Once I say “no, lower”, you start backing down by one’s until you hit the number.

1. What is the worst case number of guesses you have to make? **205 guesses if the number is 996**
2. Best case number of guesses? **1 guess if the number is 0**
3. Average case (say we play the game 25,000 times and average the # of guesses)? **Accept 102, 103 for full credit, 101 partial credit**
4. If we generalize the multiple-of-5 approach to guess a number between 0 and N (where N is a multiple of 5) what is the worst case number of guesses (expressed as “Big-Oh” notation)?

 **O(N/5) is O(N)**

1. Is this multiple-of-5 approach better, worse, or the same as simply starting with 0 and guessing numbers in order, increasing by one each time? Give me a theoretical answer, and a practical answer.

 **Theoretical analysis: guessing each number in order is linear, O( ½ N) which is O(N). The skip-5 approach is also O(N) (part d). So both are the same complexity theoretically.**

 **Practical analysis: The factor of 1/5 on the skip-5 approach does give smaller run times than the factor of ½ on the every number approach. A person buying hardware, for example, to support this algorithm (like eBay or Google has to do) would need 2 times as much equipment to support the every-number approach.**